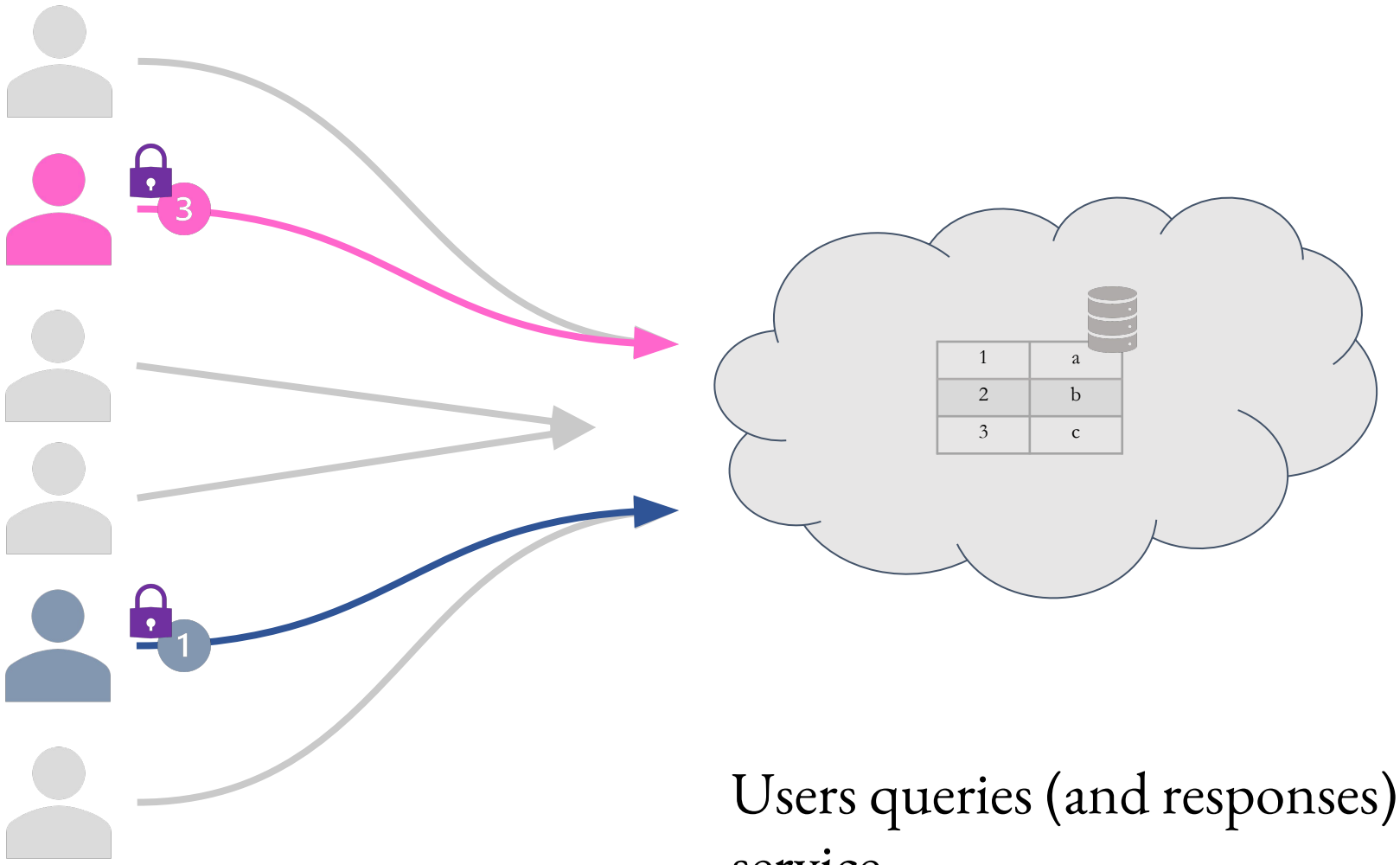
Batched Differentially Private Information Retrieval

Kinan Dak Albab*, Brown University
Rawane Issa*, Mayank Varia, Boston University
Kalman Graffi Honda Research Institutes (EU)

Private Information Retrieval (PIR)



Users queries (and responses) are hidden from the service.

Private Information Retrieval (PIR)

- First single server [KO97] and multi server [CKGS98] protocols
- PIR with *preprocessing* and sublinear *online* work [BIM04]
- Private presence discovery [BDG15]
- Anonymous messaging [MOTBG11][AS16]
- Certificate transparency [LG15]
- Safe browsing [KC21]

Private Information Retrieval (PIR)

- First single server [KO97] and multi server [CKGS98] protocols
- PIR with *preprocessing* and sublinear *online* work [BIM04]
- Private presence discovery [BDG15]
- Anonymous messaging [MOTBG11][AS16]
- Certificate transparency [LG15]
- Safe browsing [KC21]

Real-world challenge: Scale

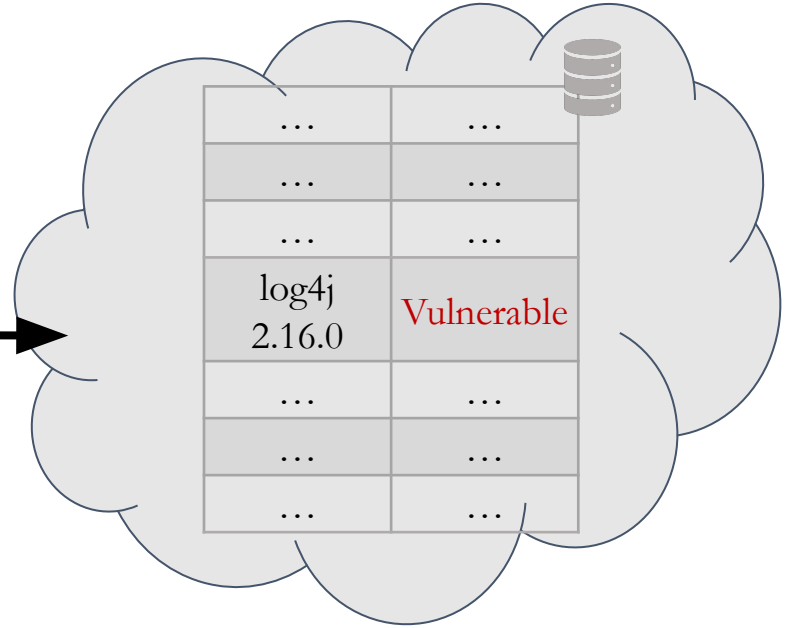
Example Application: Software Dependencies

Open

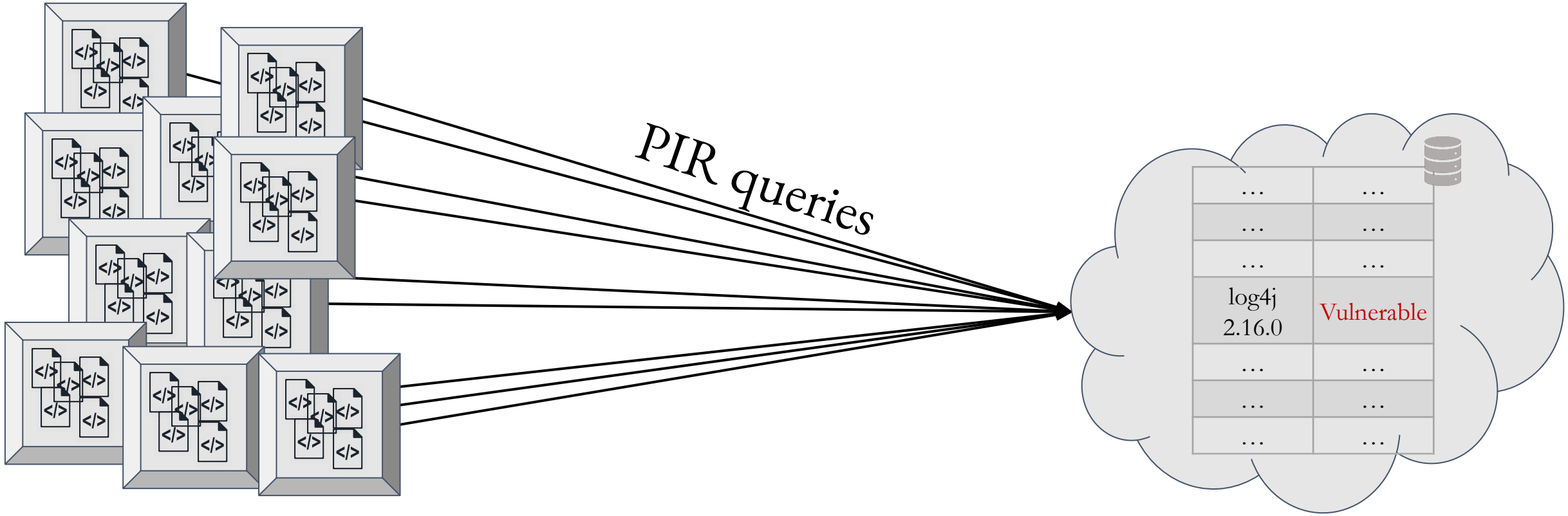
pom.xml
~/Desktop

```
39    <dependencies>
40      <dependency>
41        <groupId>com.puppycrawl.tools</groupId>
42        <artifactId>checkstyle</artifactId>
43        <version>7.7.1</version>
44      </dependency>
45      <dependency>
46        <groupId>org.jdom</groupId>
47        <artifactId>jdom</artifactId>
48        <version>1.1</version>
49      </dependency>
50      <dependency>
51          <groupId>org.apache.logging.log4j</groupId>
52          <artifactId>log4j-core</artifactId>
53          <version>2.16.0</version>
54      </dependency>
55      <dependency>
56        <groupId>com.google.collections</groupId>
57        <artifactId>google-collections</artifactId>
58        <version>1.0</version>
59      </dependency>
60      <dependency>
61        <groupId>org.slf4j</groupId>
62        <artifactId>slf4j-api</artifactId>
63        <version>1.7.25</version>
```

PIR



Example Application: Software Dependencies



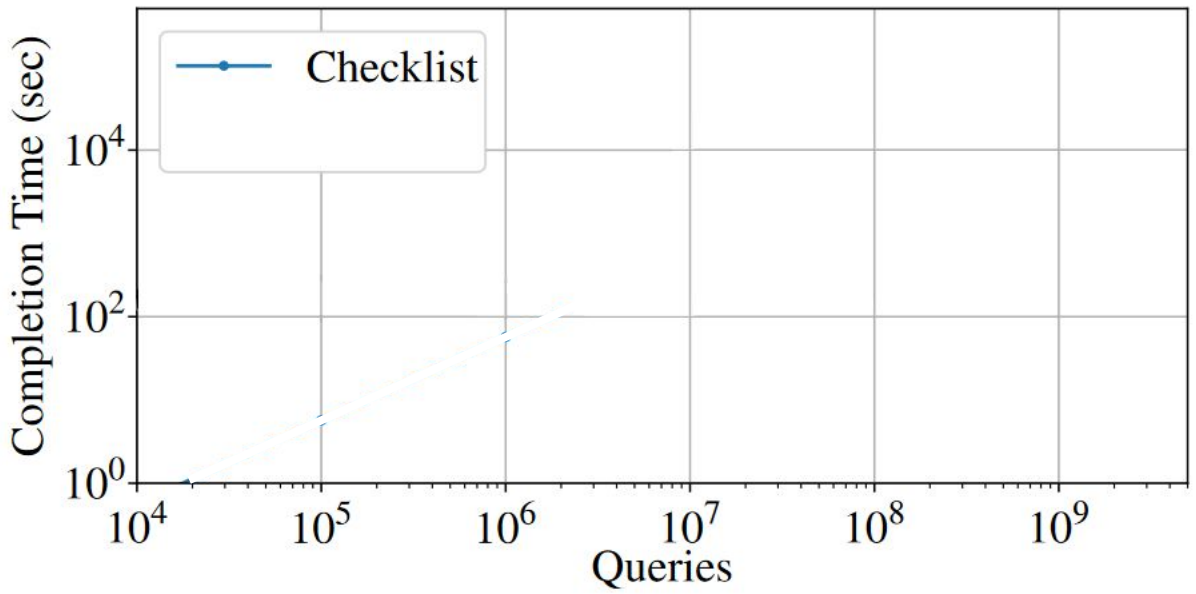
100M Repositories
× 10s of dependencies each

= Billions of queries!

database with
millions of elements

Do PIR constructions scale to our applications?

Checklist [CK21]



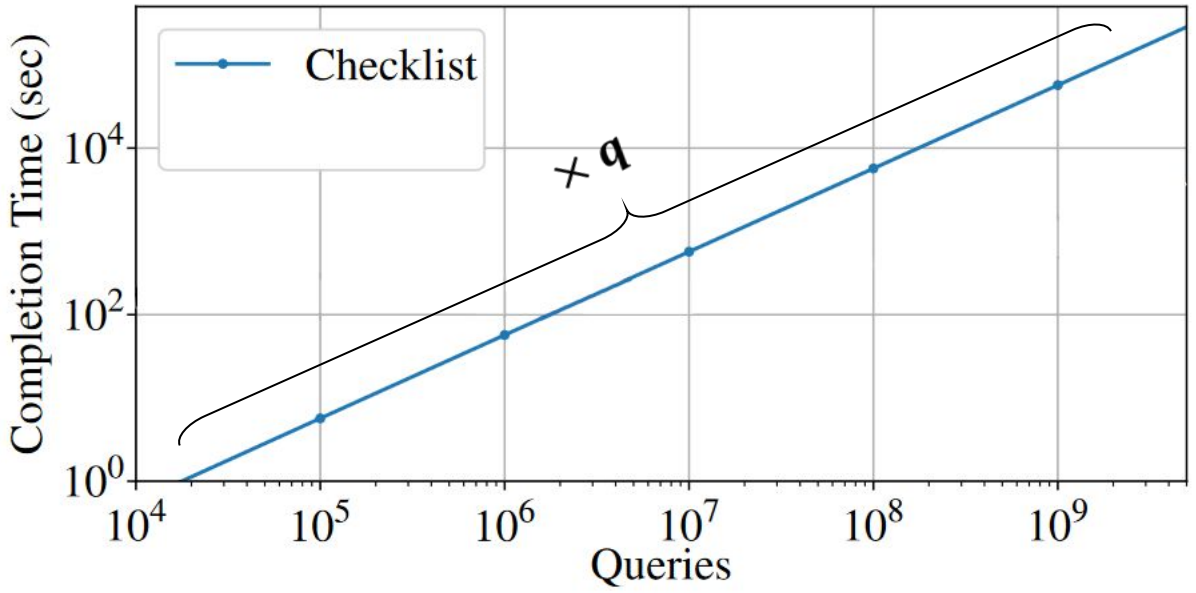
2.5M database

Legend

n is the size of the database
q is the number of incoming queries

Do PIR constructions scale to our applications?

Checklist [CK21]



2.5M database

Per-query Computation: $O(\sqrt{n})$

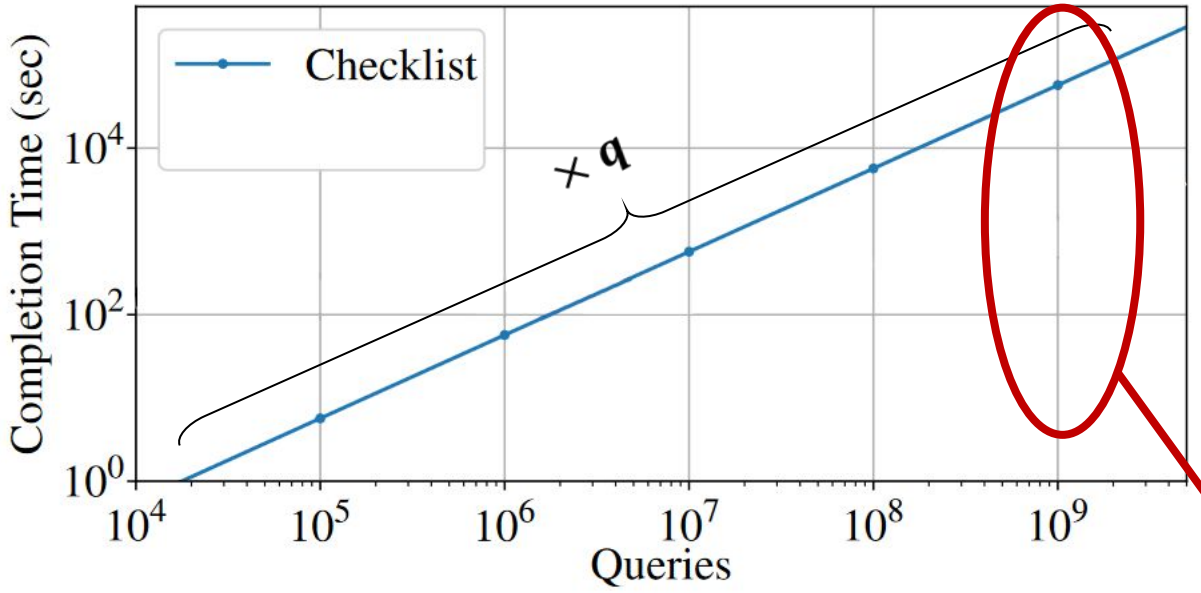
Total Computation: $O(q \times \sqrt{n})$

Legend

n is the size of the database
q is the number of incoming queries

Do PIR constructions scale to our applications?

Checklist [CK21]



2.5M database

Per-query Computation: $O(\sqrt{n})$

Total Computation: $O(q \times \sqrt{n})$

Software Dependencies

Legend

n is the size of the database
q is the number of incoming queries

How can we scale PIR further?

- Limited further improvements for per-query costs (due to theoretical lower bounds)

- Opportunity: amortize PIR's overhead over queries
  - handle queries in batches

- Challenge:
  - queries can be made by different users
  - amortize while performing expensive operations offline

Our contribution

DP-PIR: Novel differentially private PIR protocol geared towards applications with <u>many</u> queries

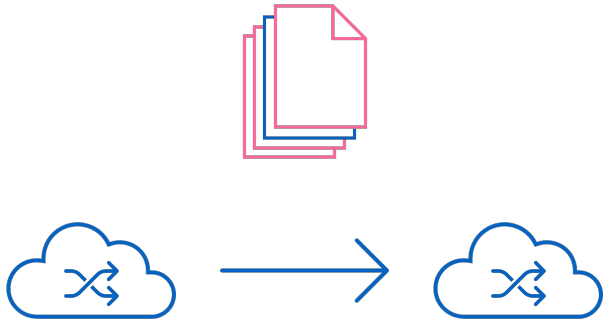
- Constant amortized complexity for servers and users
- Computation in $O(q + n)$ (rather than $O(q \times \sqrt{n})$
- Novel secret sharing scheme → online protocol only uses cheap arithmetic operations

DP-PIR Overview

Differential Private Leakage

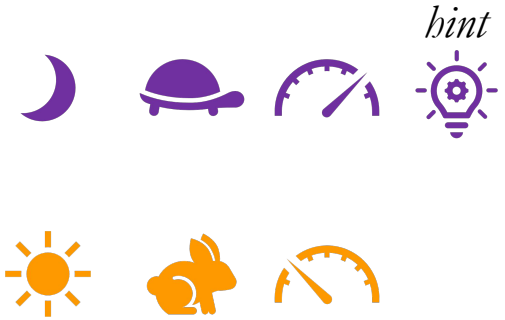
Noisy Access Patterns

Batching from different client



Mixnet

Offline/Online Staging



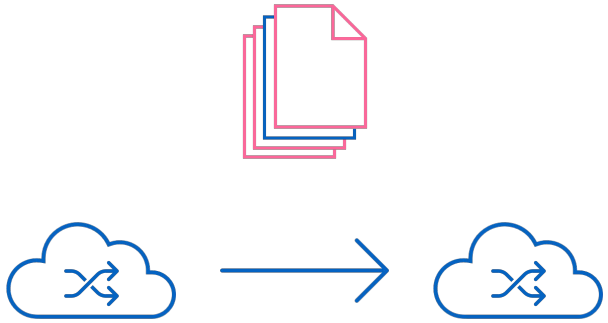
PK operations Offline

Simple construction: everything online!

Differential Private Leakage

Batching from different client

Offline/Online Staging

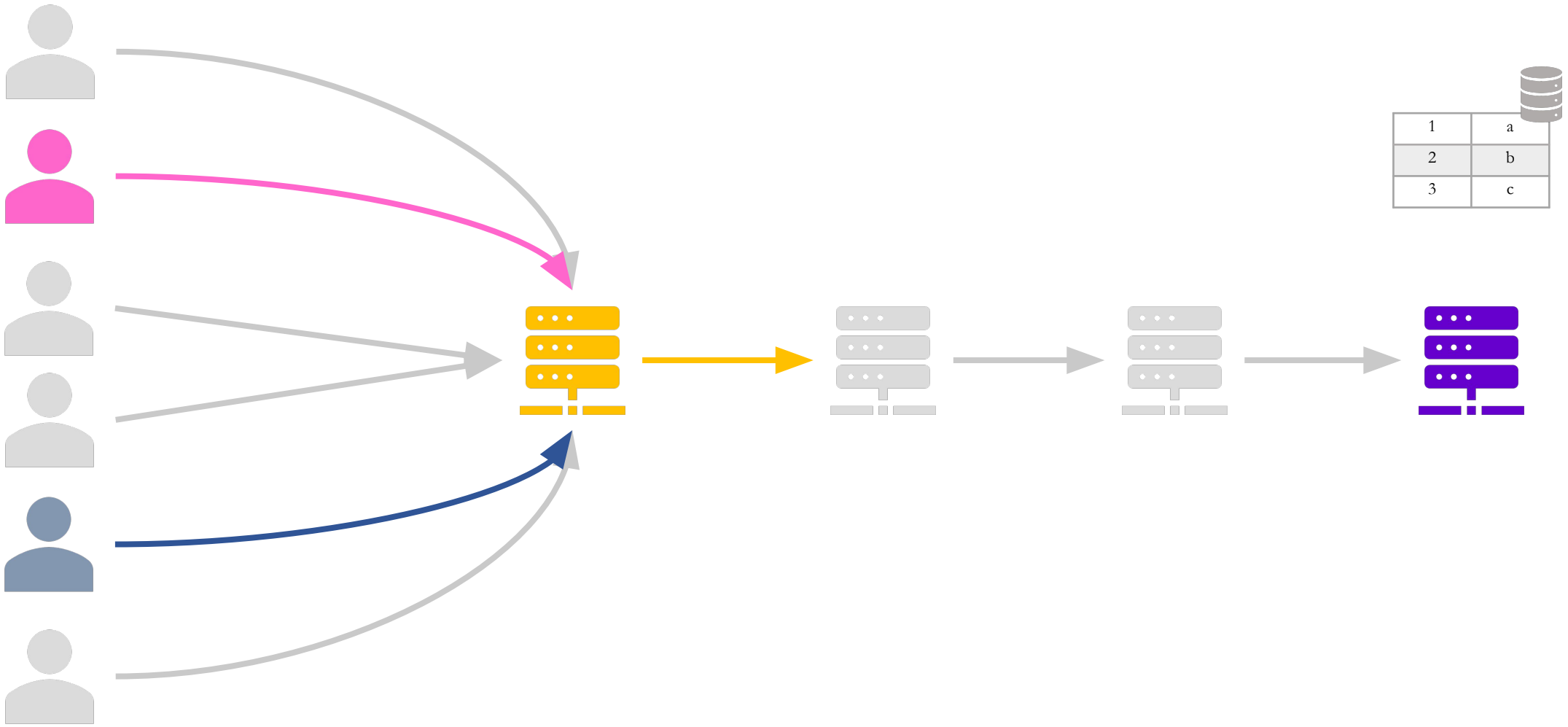


Noisy Access Patterns

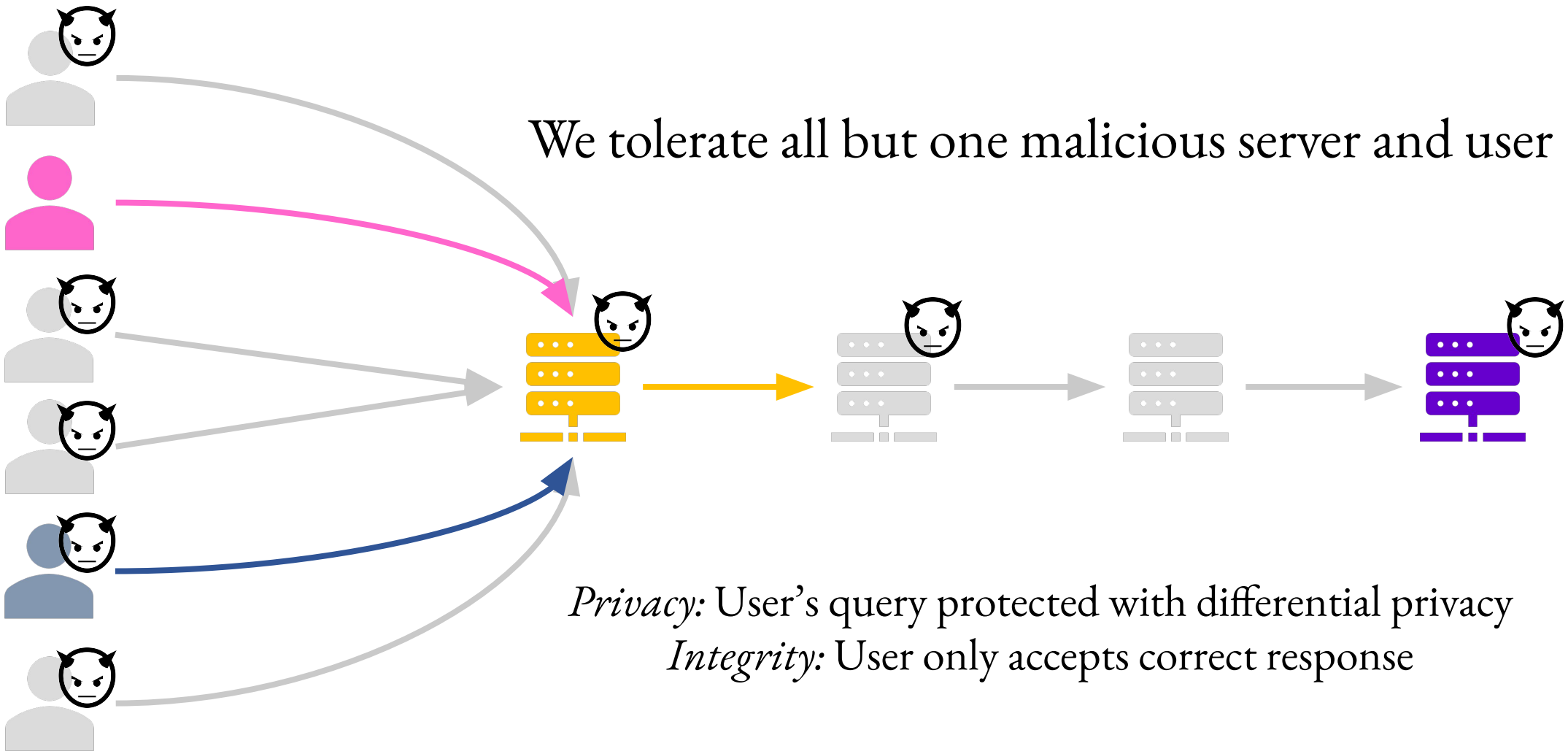
Mixnet

PK operations Offline

Our setup: multiple servers in a chain



Threat Model

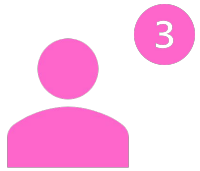


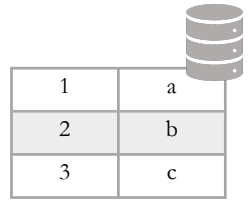
We tolerate all but one malicious server and user

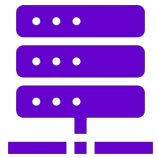
Privacy: User's query protected with differential privacy

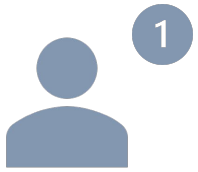
Integrity: User only accepts correct response

Two clients and two servers

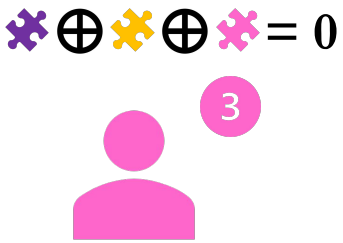


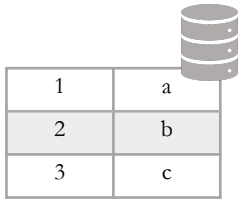


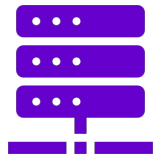


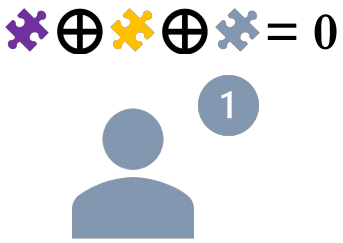


1. Users sample masks for later use

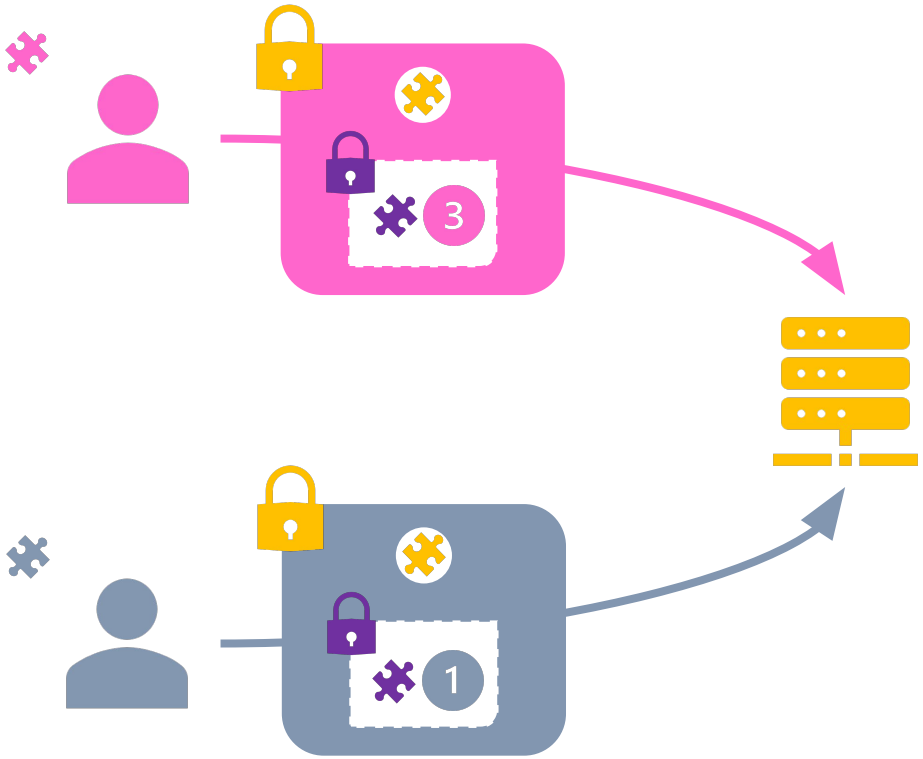


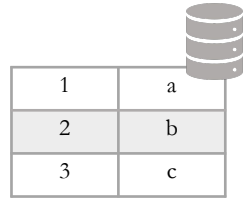


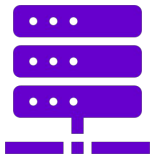




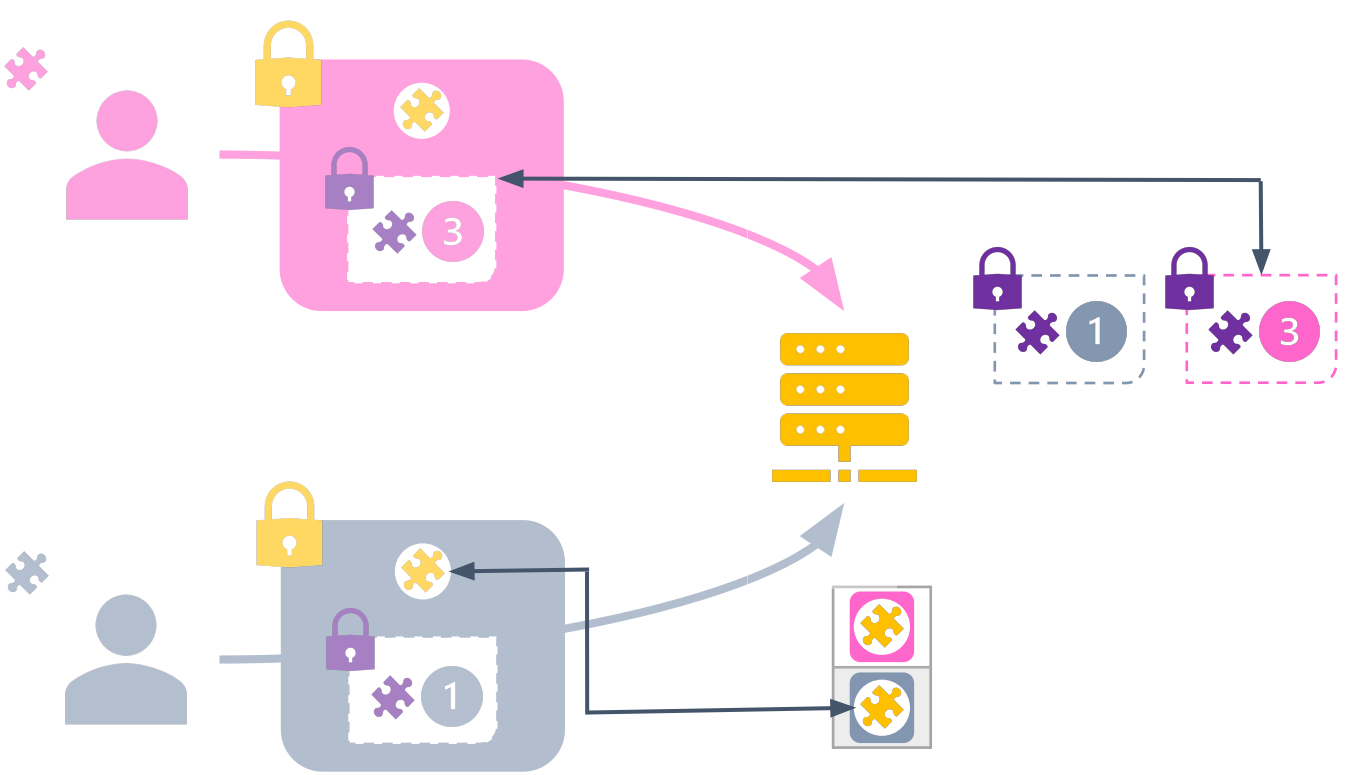
2. Users onion encrypt query and masks

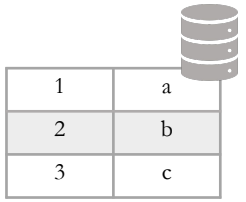


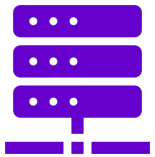




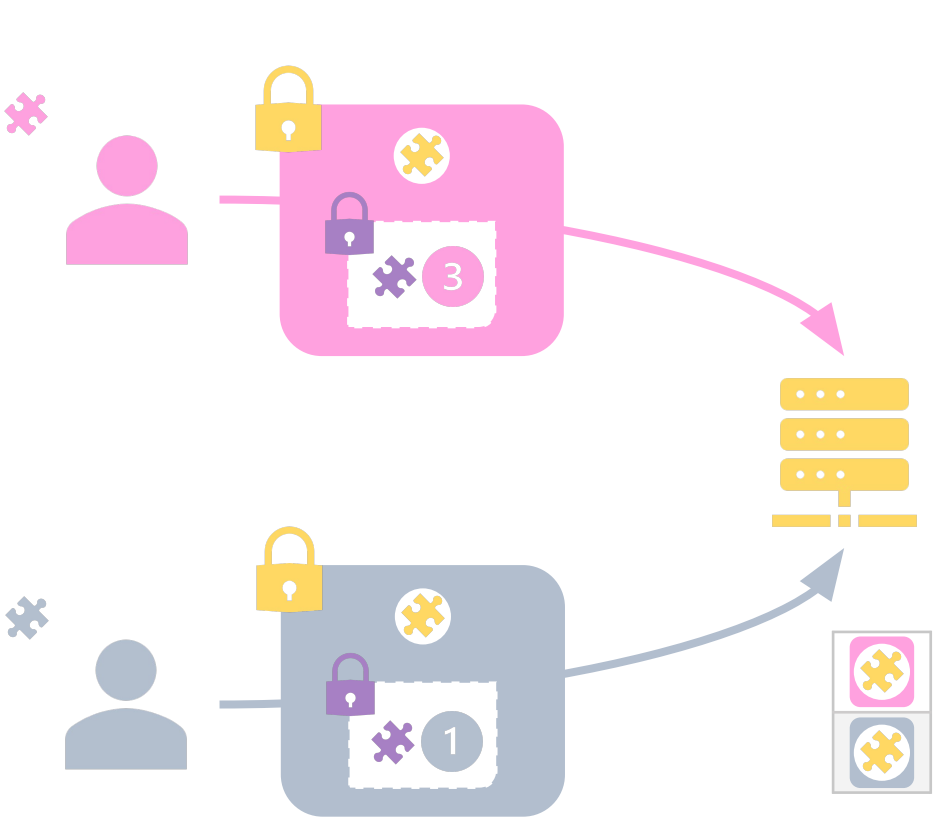
3. Server decrypts outer layer

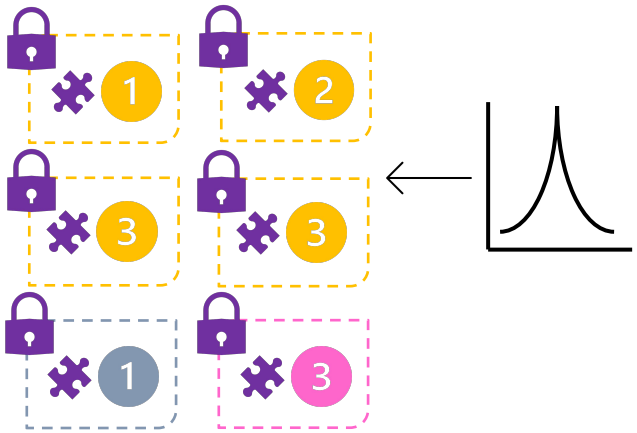


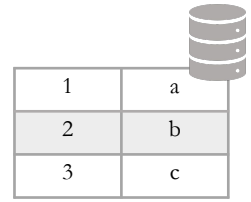


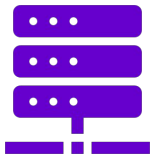


4. Server samples DP noise queries

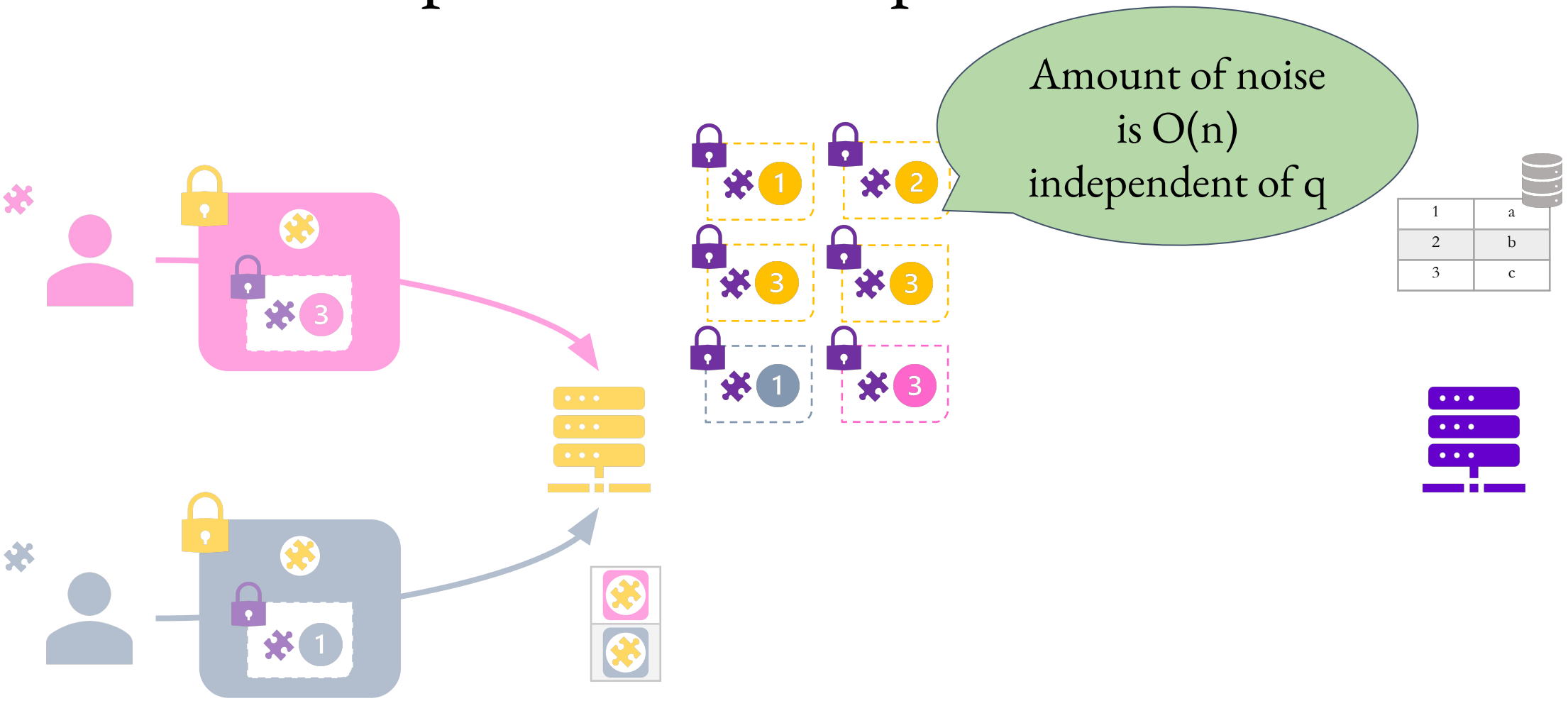




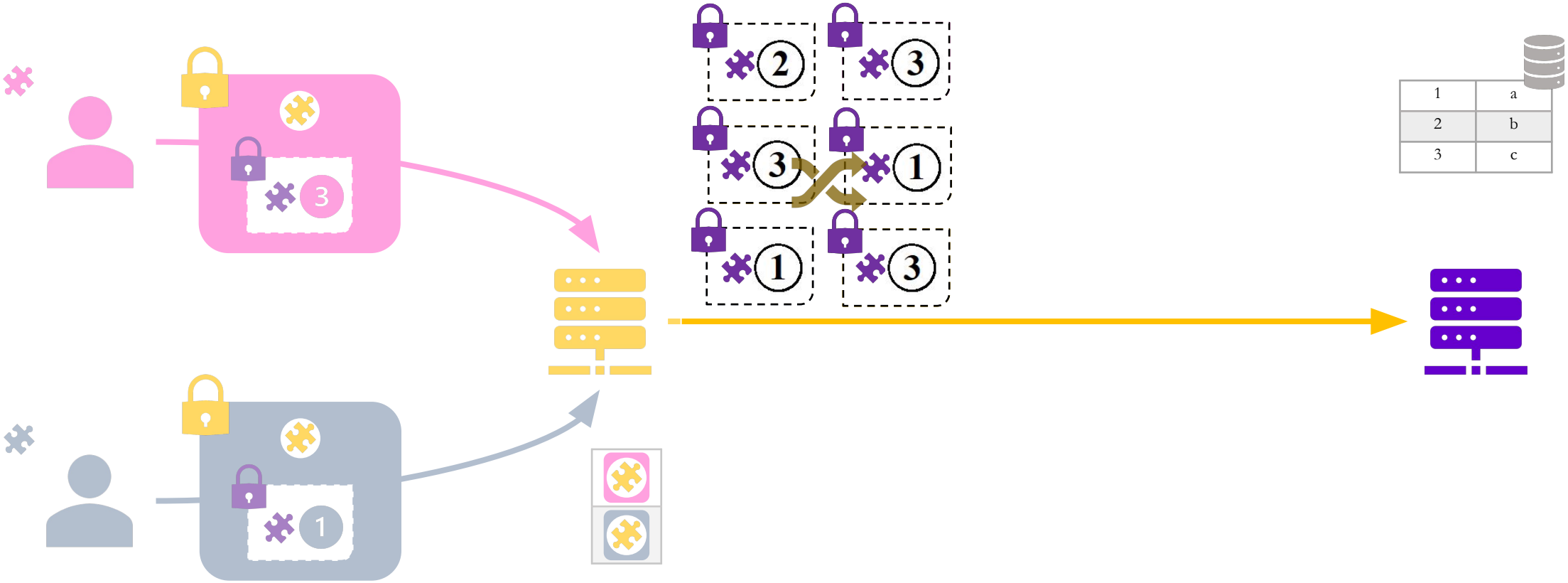




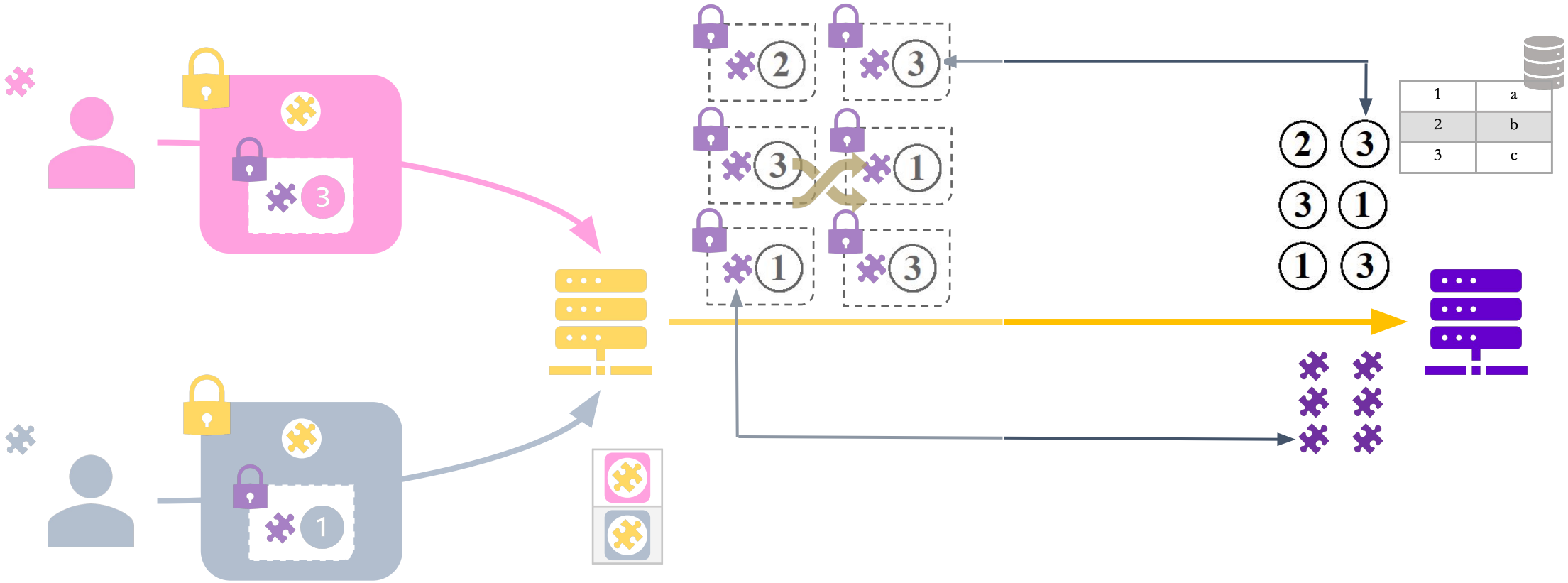
4. Server samples DP noise queries



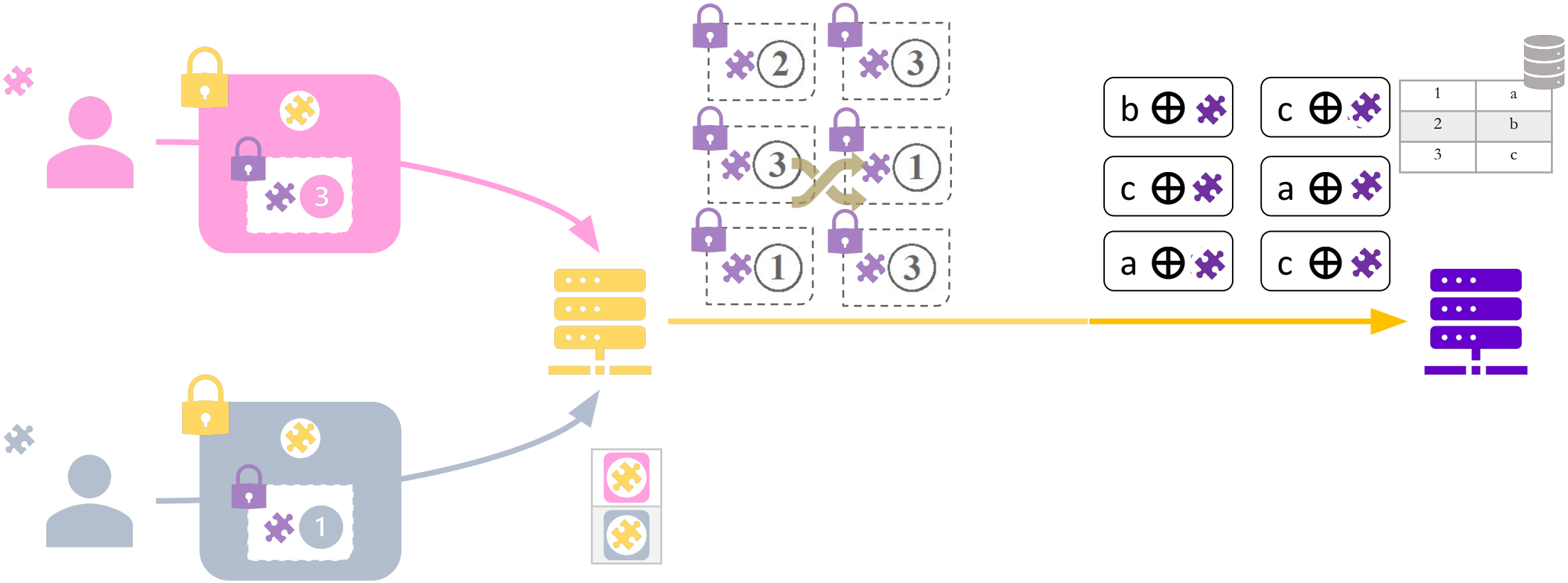
5. Server shuffles real and noise queries



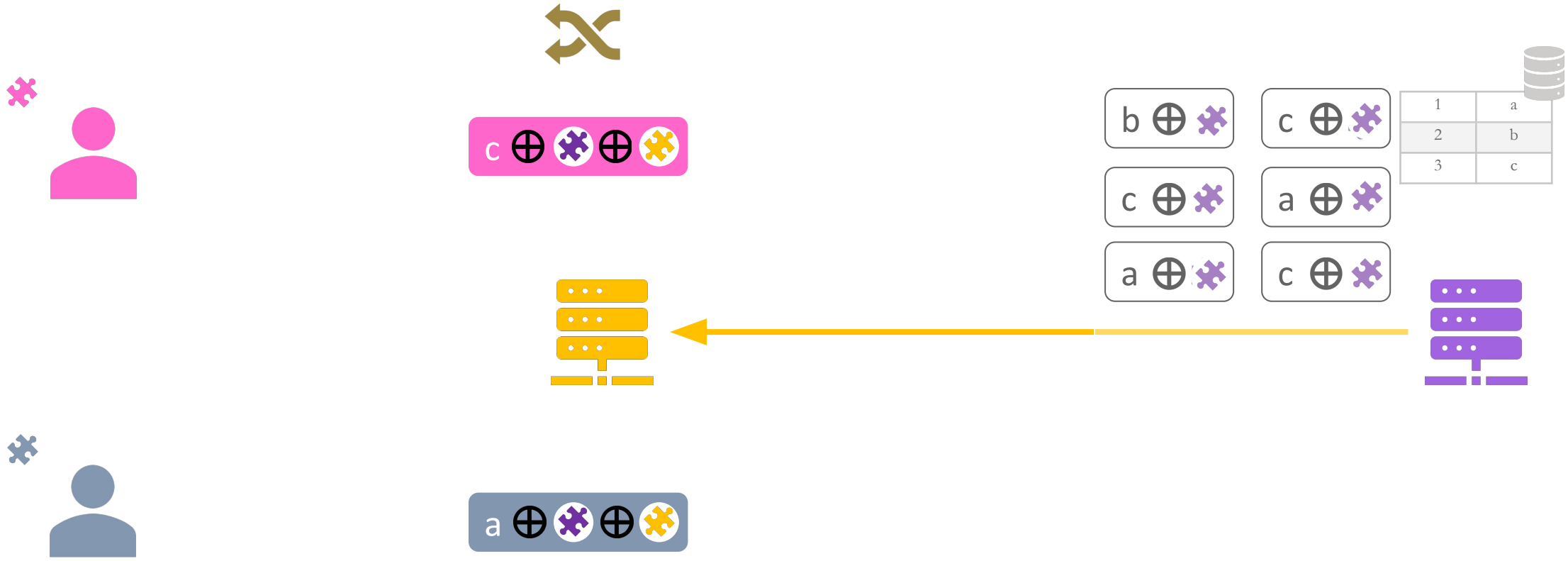
6. Last server decrypts queries



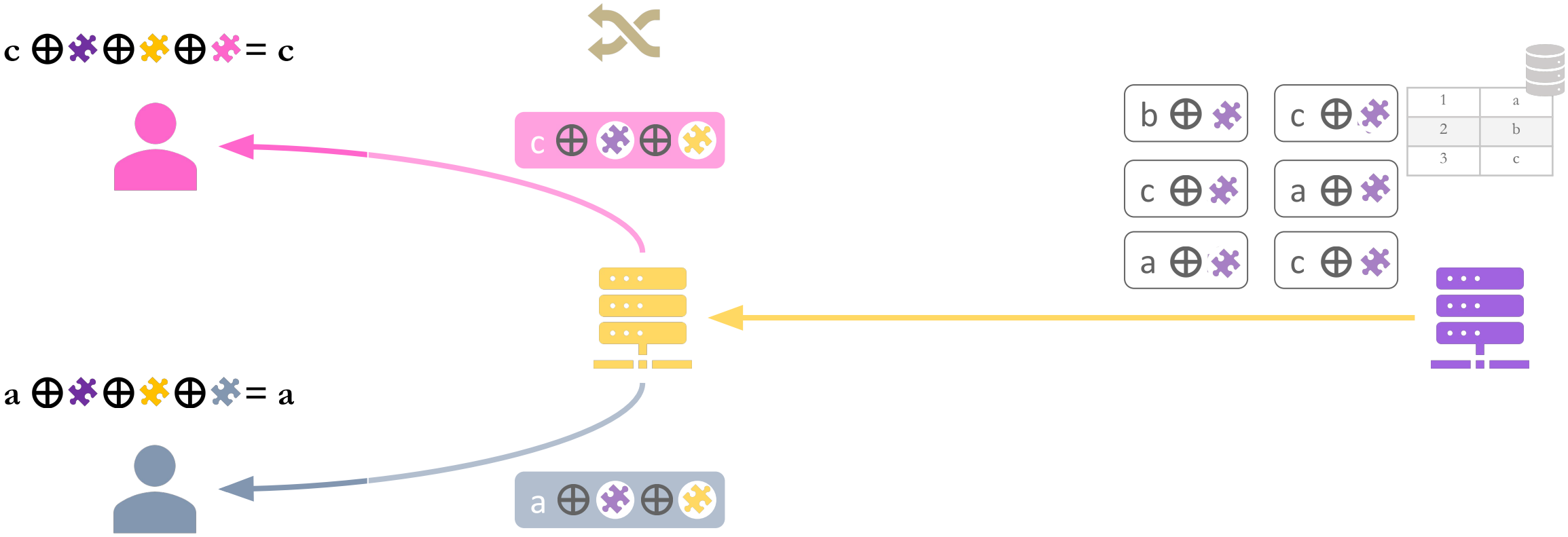
7. Last server finds responses and applies masks



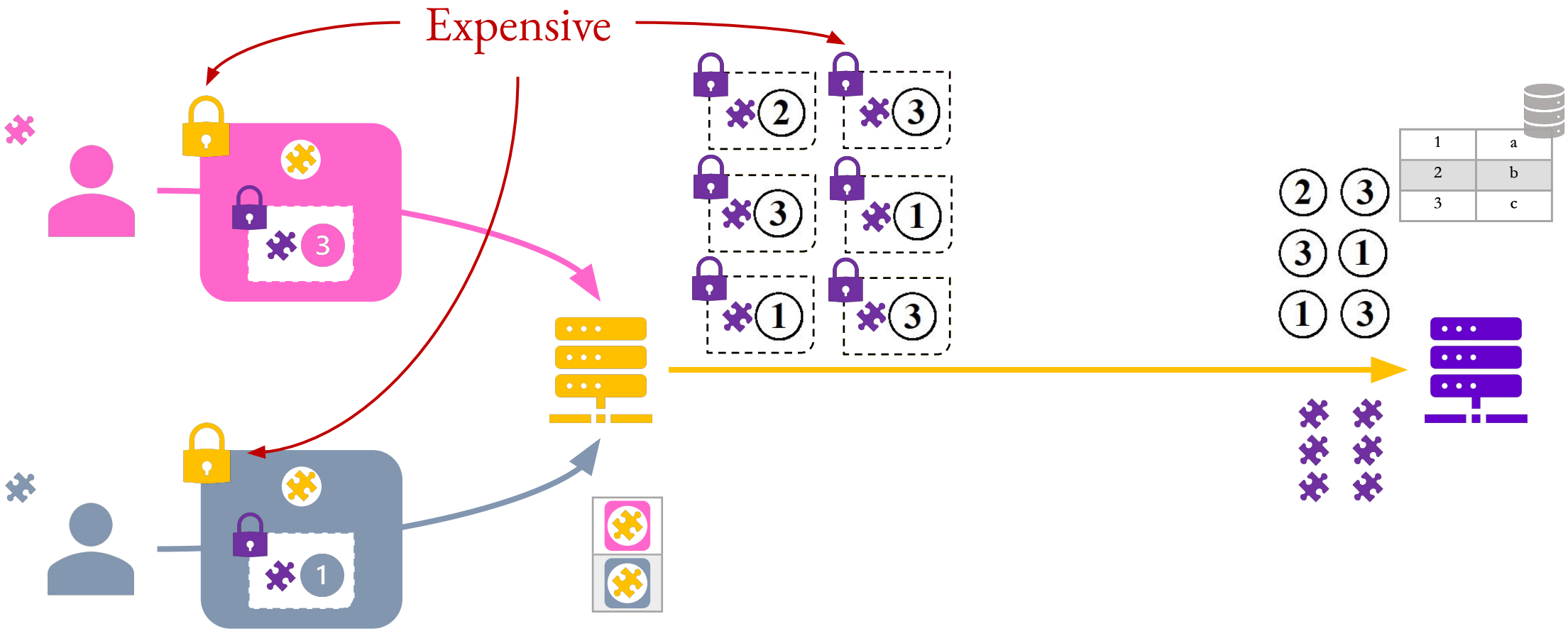
8. Server deshuffles and masks responses



9. Users reconstruct responses



Challenge: expensive public key operations



Solution: move public key operations offline

- We use cheap secret sharing instead of onion encryption online

- We use onion encryption offline to install secret shares at each server
  - servers use those secret shares online

Secret sharing scheme

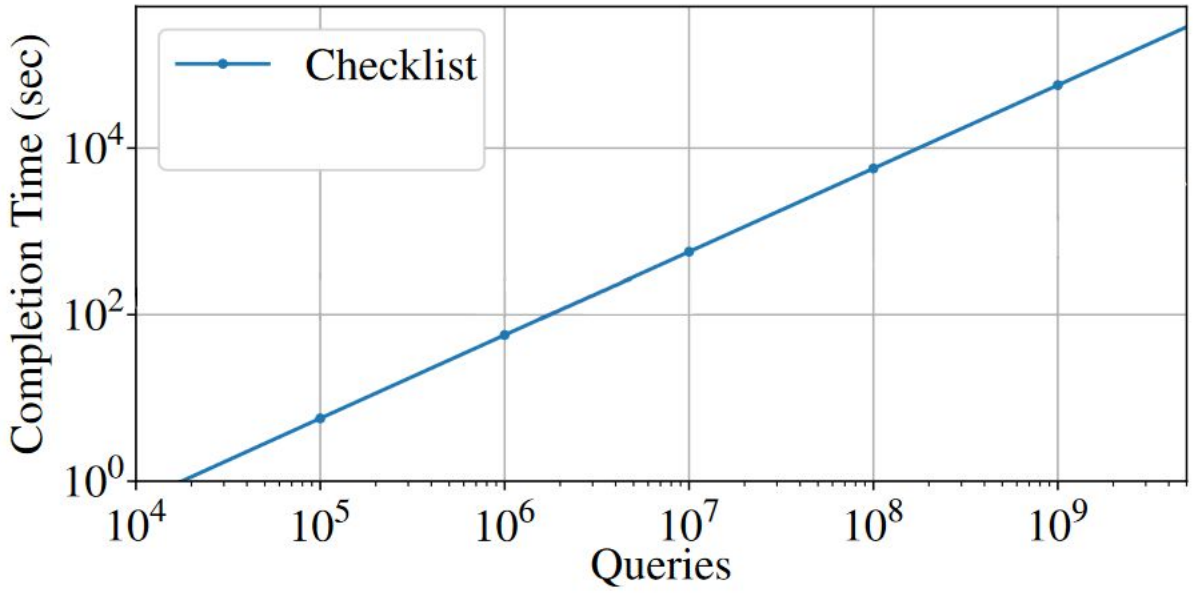
- Secret sharing scheme needs similar properties to onion encryption:
  - Incremental: query is reconstructed one server at a time in the chain

    (≈≈ Onion)
  - Non-malleable: tolerate adversary that maliciously modifies shares

    (≈≈ CCA-security)

- First secret sharing scheme that is both *incremental* and *non-malleable*!

DP-PIR scales to many queries!



2.5M database

Total online computation:

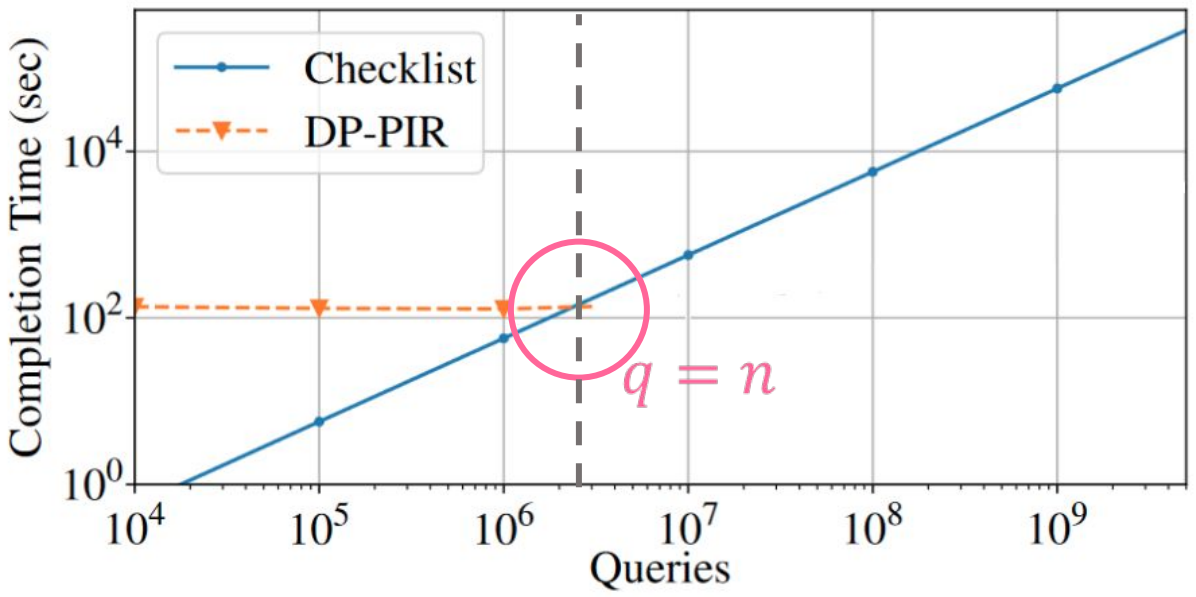
Checklist: $O(q \times \sqrt{n})$

DP-PIR: $O(q + n)$

Legend

n is the size of the database
q is the number of incoming queries

DP-PIR scales to many queries!



2.5M database
DP-PIR with epsilon = 0.1, delta = 10^-6

Work per query when $q \ll n$

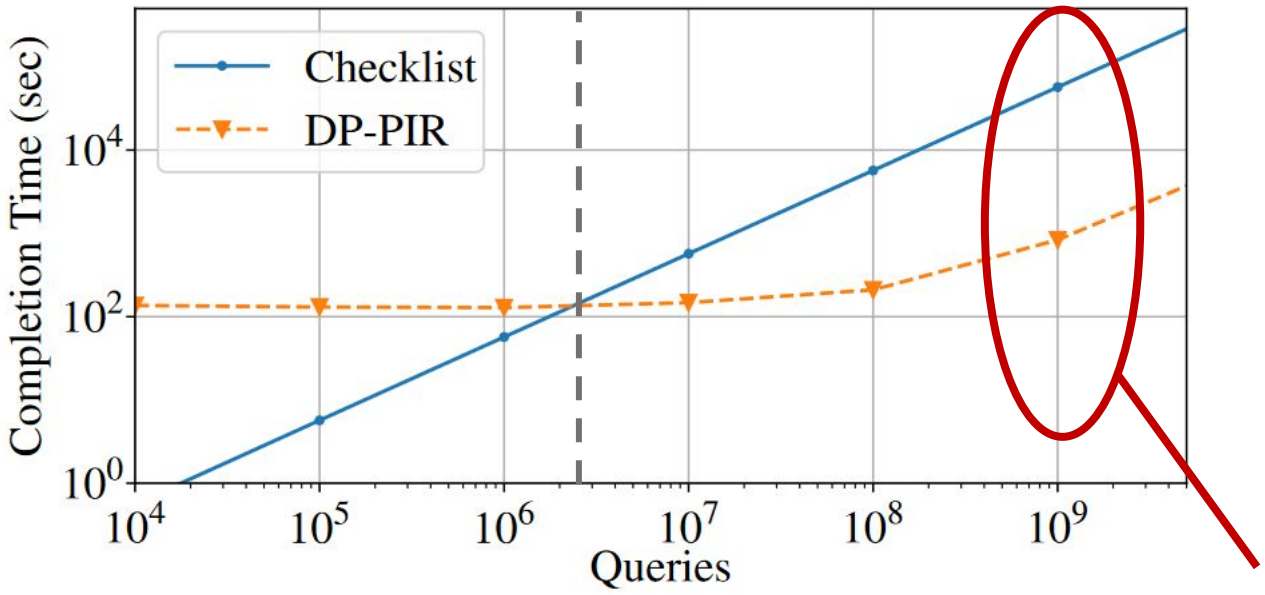
Checklist: $O(\frac{q \times \sqrt{n}}{q})$ per query: $O(\sqrt{n})$

DP-PIR: $O(\frac{q+n}{q})$ per query: $O(n)$

Legend

n is the size of the database
q is the number of incoming queries

DP-PIR scales to many queries!



2.5M database
DP-PIR with epsilon = 0.1, delta = 10^-6

Work per query when $q \geq n$

Checklist: $O(\frac{q \times \sqrt{n}}{q})$ per query: $O(\sqrt{n})$

DP-PIR: $O(\frac{q+n}{q})$ per query: $O(1)$

2 orders of magnitude speedup

Legend

n is the size of the database
q is the number of incoming queries

In the paper

- Other applications: mobile App Stores
- More experiments:
  - Comparison to other PIR protocols
  - Batch size → speedup over other protocols, latency
  - …
- Formalization of incremental non-malleable Secret Sharing properties

Conclusion

DP-PIR: A novel differentially private PIR protocol that scales to many queries:

- Constant amortized complexity for servers and users
- Enables applications of PIR that were previously impractical

Extended paper with details, proofs, and more experiments on eprint 2020/1596

Code on github: https://github.com/multiparty/DP-PIR

Thank you!

babman@brown.edu